COMP3234B Computer and Communication Networks
ELEC3443B Computer Networks

Programming Assignment

Total 12 points

Due date: 17:00 April 12, 2023

Hand in the assignment via the Moodle System.

Overview

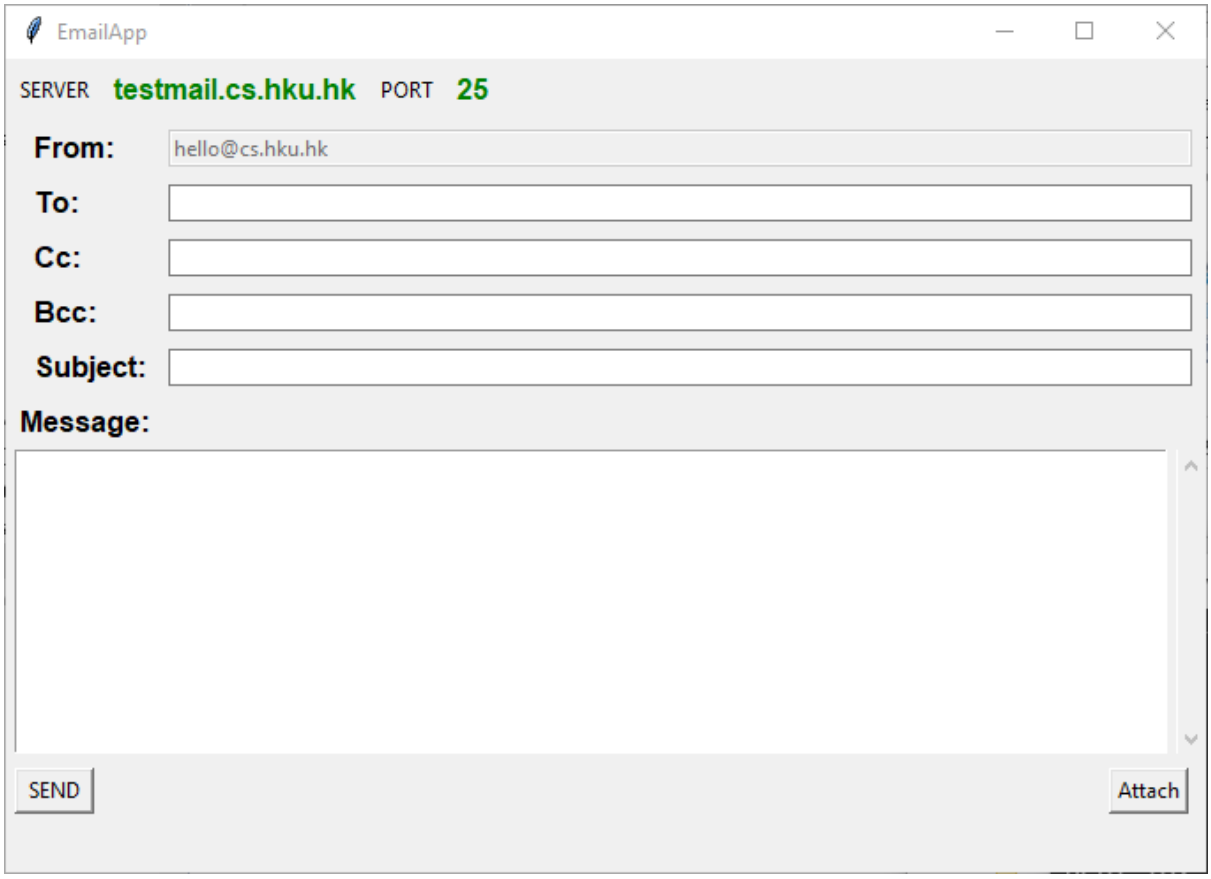
In this assignment, you are going to implement an Email client that allows a user to send emails with an attachment to a target group of recipients via the department's SMTP server – **testmail.cs.hku.hk**. If an attachment file is included, the program generates a MIME mail message encapsulating the text message and a base64 encoded attachment in the message body. If no attachment file is included, the program simply encapsulates the text message in the message body. The program communicates with a standard SMTP server (testmail.cs.hku.hk port 25) to send the mail to a group of recipients. When the recipient receives and views the email, the email message should be recognized and displayed correctly by any standard email client with the sender, receiver, subject header information, the text message as the mail body, and the decoded file attachment (if included).

Objectives

1. An assessment task related to ILO4 [Implementation] – "be able to demonstrate knowledge in using Socket Interface to design and implement a network application".
2. A learning activity to support ILO1, ILO2, & ILO4.
3. The goals of this programming assignment are:
   - to get solid experience in using Socket functions to implement the SMTP protocol;
   - to get a good understanding of sending emails with attachment under the MIME messaging standard.

User Interface Design

Our email client program makes use of the Tkinter module to implement the UI for accepting all user inputs. Tkinter is the standard GUI library for Python. **You are not required** to write the UI, as the UI framework (**Email-UI.py**) will be provided to you. It consists of the necessary Tk code to draw the UI.



There are in total 5 inputs and 2 buttons:

- "To:" field – A **required** input; the user must provide **the list of recipients' email addresses** here; each email address is separated by a comma.
- "Cc:" field – An optional input; the user may provide **the list of cc recipients' email addresses** and email addresses are separated by commas.
- "Bcc:" field – An optional input; the user may provide **the list of bcc recipients' email addresses** and email addresses are separated by commas.
- "Subject:" field – A **required** input; the user must provide the Subject header of the email.
- "Message" field – A **required** input; the user must provide the text content of the email message.
- "Attach" button – The user clicks this button for selecting the attachment file.
- "Send" button – The user clicks this button for sending the email to the SMTP server.

The template file Email-UI.py contains the following utility functions for accessing individual input fields:

```
#This set of functions is for getting the user's inputs
def get_TO():
  return tofield.get()

def get_CC():
  return ccfield.get()

def get_BCC():
```

```
  return bccfield.get()

def get_Subject():
  return subjfield.get()

def get_Msg():
  return SendMsg.get(1.0, END)
```

When the user clicks on the "Attach" button, the following utility function will be invoked for selecting a file and **returning the opened file object and the filename** via two global variables.

```
#This function calls the file dialog for selecting the attachment file.
#If successful, it stores the opened file object to the global
#variable fileobj and the filename (without the path) to the global
#variable filename. It displays the filename below the Attach button.
def do_Select():
  global fileobj, filename
  if fileobj:
    fileobj.close()
  fileobj = None
  filename = ''
  filepath = filedialog.askopenfilename(parent=win)
  if (not filepath):
    return
  print(filepath)
  if sys.platform.startswith('win32'):
    filename = pathlib.PureWindowsPath(filepath).name
  else:
    filename = pathlib.PurePosixPath(filepath).name
  try:
    fileobj = open(filepath,'rb')
  except OSError as emsg:
    print('Error in open the file: %s' % str(emsg))
    fileobj = None
    filename = ''
  if (filename):
    showfile.set(filename)
  else:
    alertbox('Cannot open the selected file')
```

When the user clicks the "Send" button, the system will run the **do_Send()** function, this is the **main task** you are going to work on for this assignment.

Specification of the Email client program

1. **Rename** the UI template file Email-UI.py to EmailApp.py
2. **Change** the two global variables **YOUREMAIL** and **MARKER** to store your CS email address and HKU Student number.

```
# Replace this variable with your CS email address
YOUREMAIL = "hello@cs.hku.hk"
# Replace this variable with your student number
MARKER = '3035999999'
```

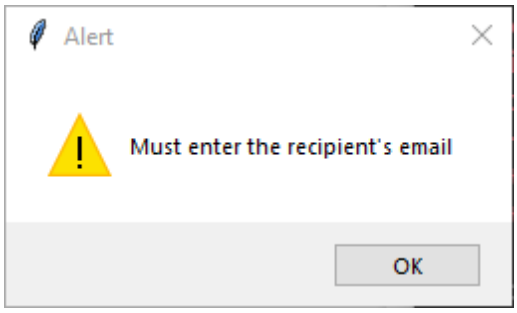
3. **Implement** the do_Send() function, which contains the main logic of this email client. It involves three major tasks.

   Task 1 – Check all input fields.

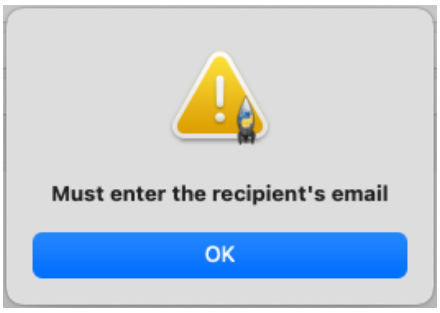
   1. The user must provide inputs to the "To:", "Subject:", and "Message:" fields.

   If the corresponding field is empty, the program should use the utility function **alertbox()** to display an alert message to the user. For example, when the "To:" field is empty, the program displays:

Windows



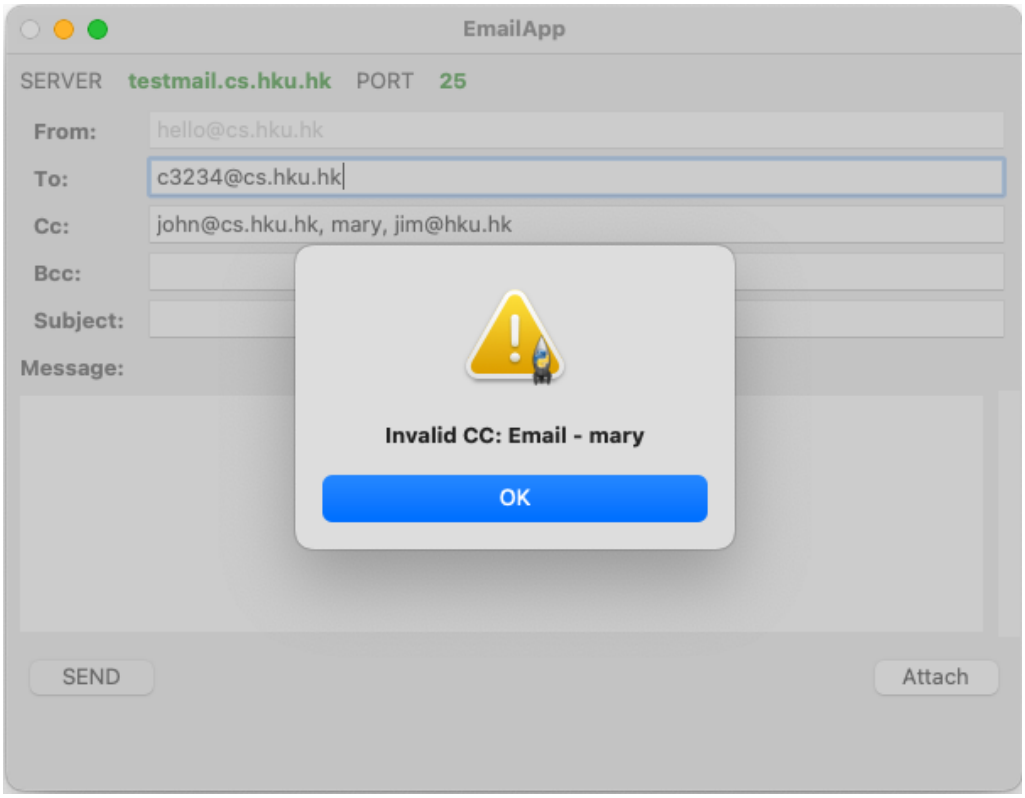
Mac



2. For the "To:", "Cc:", and "Bcc:" fields, the program expects the user to enter a list of recipients' email addresses. The program should **parse the list and check whether each address is a valid email address**. You can use the utility program **echeck()** to validate an email address.

```
#This function checks whether the input is a valid email
def echeck(email):
  regex = '^([A-Za-z0-9]+[.\-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+'
  if(re.fullmatch(regex,email)):
    return True
  else:
    return False
```

   If the address is not a valid email address, use the alertbox() function to display an alert message to the user. For example, when one of the email addresses in the cc list is not a valid address, the program displays:

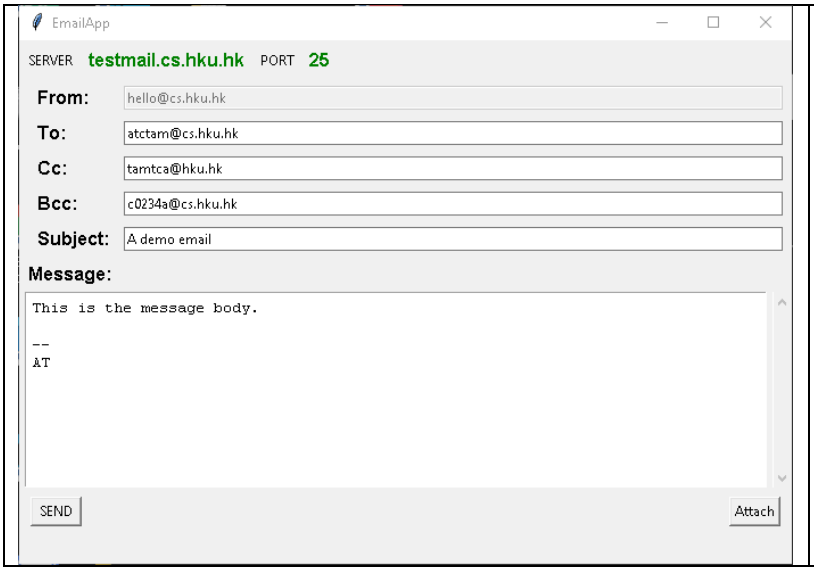


Task 2 – Compose the email message

The program should follow the RFC2822 standard (https://datatracker.ietf.org/doc/html/rfc2822) to compose the email message. A message is comprised of characters with values in the range 1 through 127 and interpreted as US-ASCII characters. A message consists of the Headers part and the Body part, which are separated by an empty line. The Headers part has multiple lines of characters; each line consists of a header field, followed by a colon, and followed by a field content. **Each line is terminated by CRLF (i.e., "\r\n").** For simplicity, you can assume that the user will not enter any non-ascii characters in the Subject and the Message fields.

1. For the Headers part, it **must include** the "From:", "To:", & "Subject:" headers. If the user has provided input in the "Cc:" field, including the "Cc:" header too. The contents should be the same as appeared in the corresponding input fields.

2. For the Body part, if the user **did not select** the attachment file, the program just copies the content from the "Message" field to the Body part. Below is a simple example. Again, for simplicity, you can assume that the user will not enter a line with more than 998 characters.




```
From: hello@cs.hku.hk
Subject: A demo email
To: atctam@cs.hku.hk
Cc: tamtca@hku.hk

This is the message body.

--
AT
```

3. If the user has attached a file, the program should follow the RFC2045 standard (https://datatracker.ietf.org/doc/html/rfc2045) to compose the email message.

   Reference: https://en.wikipedia.org/wiki/MIME#Multipart_messages
   We need to add at least two MIME header lines to the Headers part.
   (i) MIME-Version: 1.0
   (ii) Content-Type: multipart/mixed; **boundary=3035xxxxxx**

   For this assignment, **each student should use his/her student number as the boundary marker**. **This value should be defined and stored in the global variable MARKER**. This marker, which must not occur in any of the parts, is placed between the parts, and at the beginning and end of the body of the message.

   Since the program can only send one attachment file, the body of the message would only consist of 2 parts – the first part encapsulates the text content (from the message field), and the other part encapsulates the attached file in base64 encoding. Each part starts with the marker e.g., "--3035xxxxxx", and the last part ends with the marker e.g., "--3035xxxxx--". Each part should contain its own header lines, an empty line, and a body.

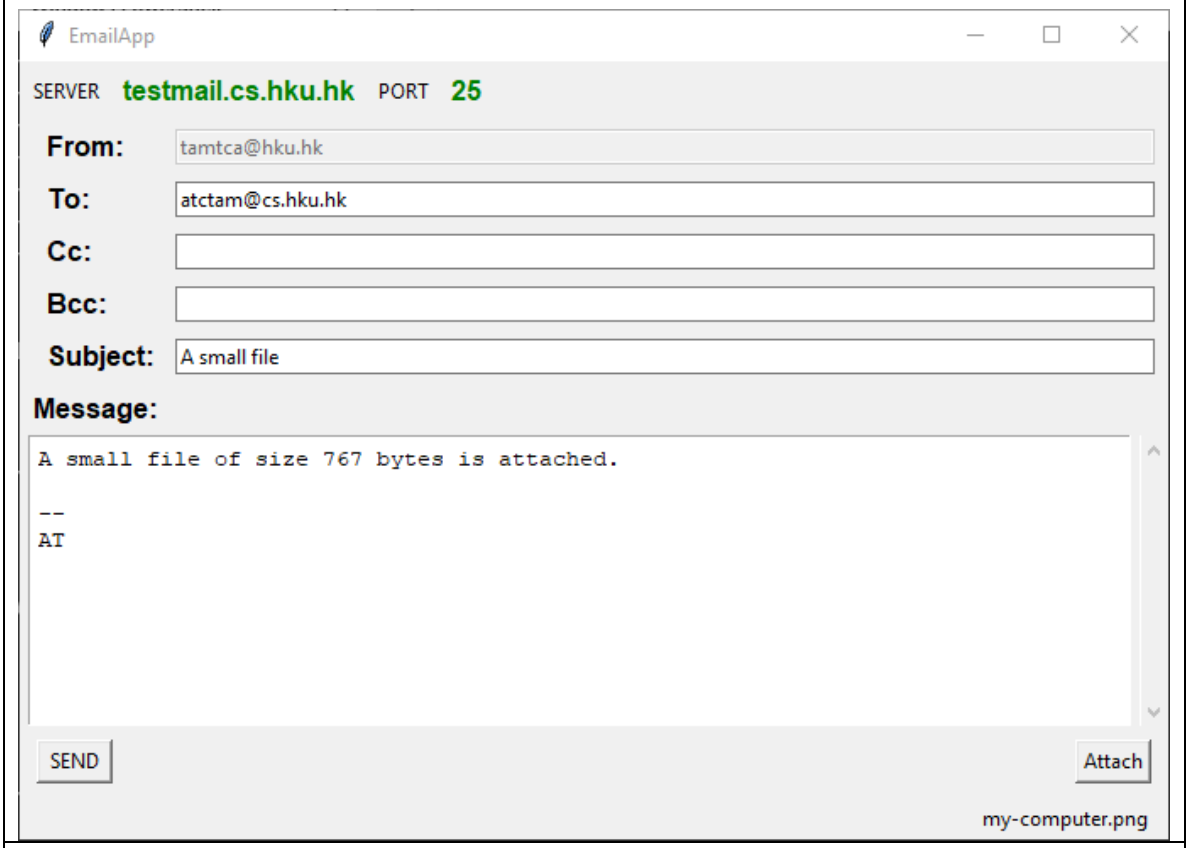
   For the text content part, it is sufficient to just add the following two headers:
   (i) Content-Type: text/plain
   (ii) Content-Transfer-Encoding: 7bit

   For the attachment part, we need to add the following three headers:
   (i) Content-Type: application/octet-stream
   (ii) Content-Transfer-Encoding: base64
   (iii) Content-Disposition: attachment; filename=*name_of_the_file*
   We can simply use the content-type application/octet-stream to represent any binary file. By using the header Content-Disposition, we tell the email user agent to store the attachment with the provided filename. Once the user has attached a file, the program can retrieve the filename via the global variable filename.

To encode the file content with Base64 encoding, we make use of the built-in python base64.py library for the purpose. One useful function is the **base64.encodebytes()** function, which encodes the file content and structures the encoded content in lines of no more than 76 characters as defined in RFC2045.

Here is an example that shows the email message with a file attachment.

Compose the email message

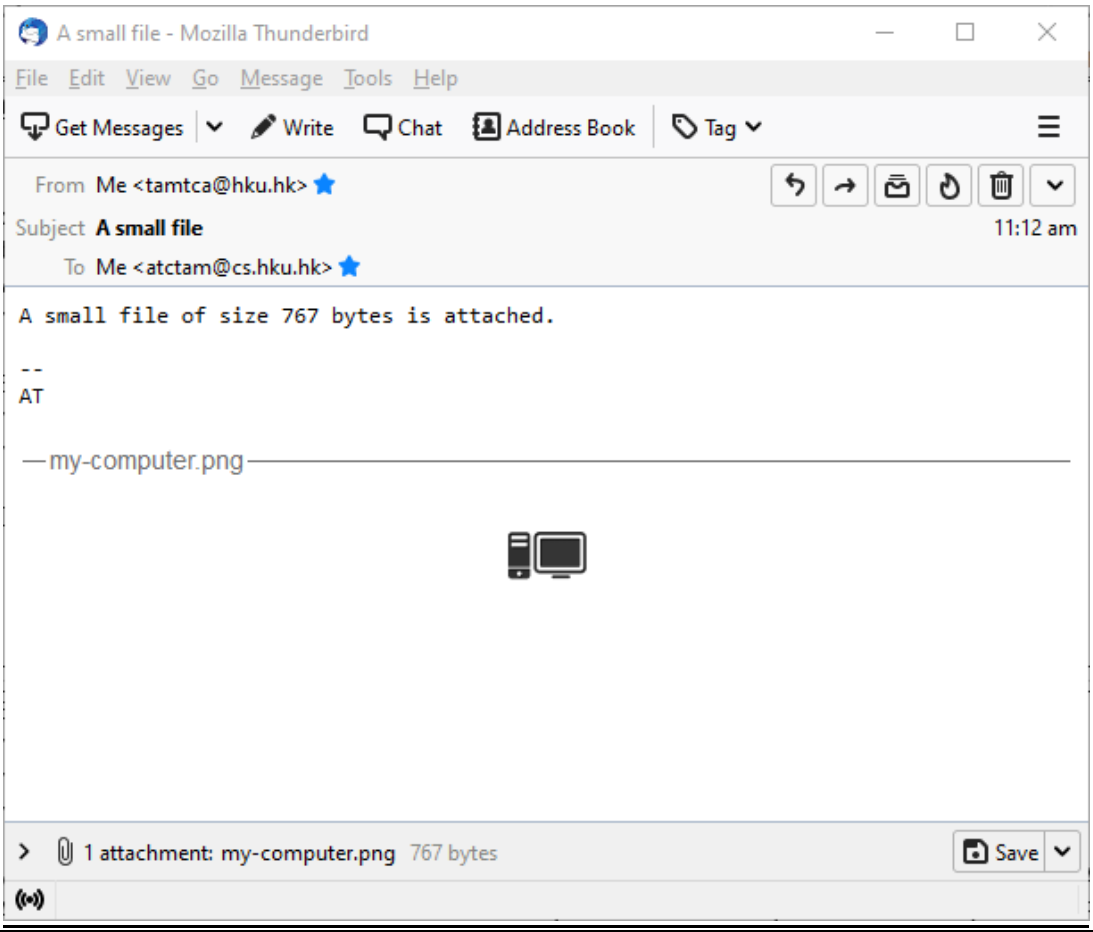


The message body is structured in MIME format

```
From: tamtca@hku.hk
Subject: A small file
To: atctam@cs.hku.hk
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=3035999999

--3035999999
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

A small file of size 767 bytes is attached.

--
AT

--3035999999
Content-Type: application/octet-stream
```

```
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=my-computer.png

iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAQAAAD9CzEMAAAAAmJLR0QA/4ePzL8AAAAJcEhZcwAA
AEgAAABIAEbJaz4AAAAJdnBBZwAAADAAAAAwAM7ujFcAAAIsSURBVFjD7ZaxaxNxFMc/75I0rRFp
xYBYaJDoIBQXHUQEBcHJyUWXQtUKgqIoOCjSP6DgWBVRcKi66uJiCApaXSwK3SzYog7VBrSKJpfr
PYdcerlffoEc1kG57/Djfu893ueX7+9xF0iUKFGiRIn+BwkOc5TNDI1ppgGYJBx2UsqVt/3XNNy
CLjFEZbxjSKfQ8yzQV5IXmble6xjF3S9jvnPIA3AHma4hEaK9nGKYeblqOSdq97teMakt8k9GQkB
Hku8NWqGyOIABecrT4BhhrrsXuWpN5eqyEZoAmArk6yLlBVXcx4+fVIW2kzspNP+Q78vVWsF9HKA
rFH2s+U5I56zKCgCkRVLTJ3wJE3AJy4Yd7Cfs5G9jyoERa2rPWYClnlnZHbQGw2IgKq52mJqARS5
Sy7SrxBz8juoCehhNz1GrhrdqtpXe8wEQB3Xcnd/bJGzFjZ0Y1EDaxuONbNIrOOdWPT3LfpHf4Ga
gGmk7S0F8Hn16QePdJPQlWq8aQdkraVLNF4YA3zwz3T7MQj86KcSAi4zYC27yBwljjEuD/RXrPaH
ZQv3Q0CdL0HqDjNcYTDYVUFfMcmYM6G5GABQp8T1EBAEqXMTjzIjtFyydyP9WHZKJk57PurrFT8E
uGRRIMM5ZjmO4AJKLUAssBBxYDuj5oxR1pKN1QBMcTIY01HyeFQQlEVedjjhQTkh5tei3w5onmNX
258Sl+d8s/cXkc1tI1tZcWOYmCjUbxP9KdwH5CIYAAAAJXRFWHRkYXRlOmNyZWF0ZQAyMDEwLTAy
LTExVDEyOjUwOjE4LTA2OjAwp3AJqwAAACV0RVh0ZGF0ZTptb2RpZnkAMjAwOS0xMC0yMlQyMzoy
Mzo1NC0wNTowMLoQUnwAAAAASUVORK5CYII=

--3035999999--
```

The resulting email message displayed by Thunderbird



Task 3 – Send the message to the SMTP server
The program makes use of the following SMTP commands for establishing the connection and sending the message.

Command	Expected Reply Code
"After TCP Connection establishment"	220
HELO or EHLO	250
MAIL FROM:	250
RCPT TO:	250
DATA	354
QUIT	221

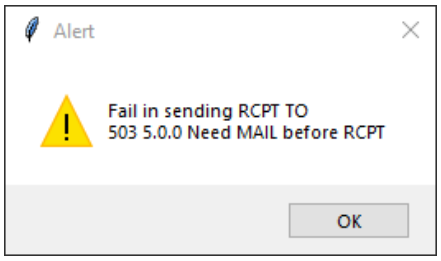
You must implement the SMTP communication using the low-level socket interface. Python has a built-in SMTP client library (smtplib), you **cannot use** this library to send the email message for this assignment. **The program should print all the transmitted commands and the received responses to the terminal/console**; however, you do not need to output the message content to the terminal. Here is the output of the previous example (not showing the message content).

```
220 testmail.cs.hku.hk ESMTP Sendmail 8.15.2/8.15.2/Debian-22ubuntu3; Fri,
10 Jun 2022 20:22:58 +0800; (No UCE/UBE) logging access from:
atctampc.cs.hku.hk(OK)-atctampc.cs.hku.hk [147.8.175.181]

EHLO 147.8.175.181

250-testmail.cs.hku.hk Hello atctampc.cs.hku.hk [147.8.175.181], pleased to
meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-EXPN
250-VERB
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH LOGIN PLAIN
250-DELIVERBY
250 HELP

MAIL FROM: <tamtca@hku.hk>

250 2.1.0 <tamtca@hku.hk>... Sender ok

RCPT TO: <atctam@cs.hku.hk>

250 2.1.5 <atctam@cs.hku.hk>... Recipient ok

DATA

354 Enter mail, end with "." on a line by itself

250 2.0.0 25ACMwu7001194 Message accepted for delivery

QUIT
```

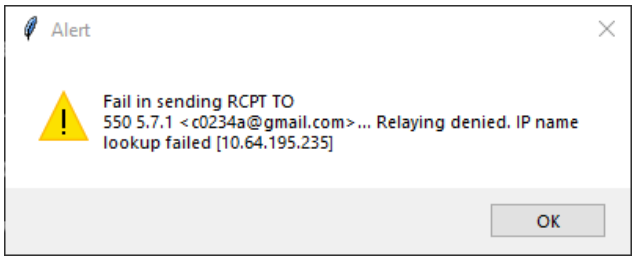
```
221 2.0.0 testmail.cs.hku.hk closing connection
```

Please note that each command must be ended with "\r\n".

The program should receive the expected reply code and the associated text content in most cases. If the program received an **unexpected response**, the program should use the alertbox() function to display an alert message together with the received response, and then **closes the TCP connection**. For example,

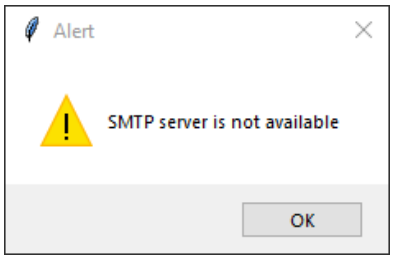


The MAIL FROM command should be sent before the RCPT TO command.



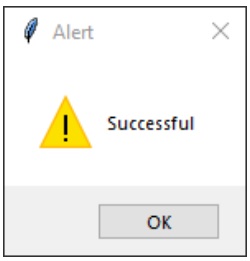
testmail.cs.hku.hk rejects the request of sending an email to a non-HKU account.

In addition, if the program could not receive any response within 10 seconds, use the alertbox() function to display an alert message and then close the TCP connection. For examples,



The server is not ready or unreachable.

When the program successfully sends the message to the server, it displays an alert message; then the whole program terminates once the user clicked the 'OK' button.



Computer Platform to Use

For this assignment, you can develop and test your email program on any platform installed with **Python 3.8 or above**.

Connect to testmail.cs.hku.hk

As this CS email server is located inside the CS network, it is **protected by the CS firewall**. To connect to testmail.cs.hku.hk, your computer needs to set up a VPN connection to HKUVPN first. Please follow the guidelines on the below HKU website for configuring and accessing HKUVPN.

https://its.hku.hk/services/network-connectivity/hkuvpn/

Submission

Name the email client program to **EmailApp.py** and submit the program to the Moodle assignment submission page on or before **April 12 at 17:00**.

The Format for the documentation

1. At the head of the submitted programs, state the
   - Student name and No.:
   - Development platform:
   - Python version:
2. Inline comments (try to be informative so that your code could be understood by others easily)

Grading Policy

As the tutors will check your source code, please write your program with good readability (i.e., with **good code conventions and comments**) so that you will not lose marks due to possible confusion.

7 points	Successfully send a simple text message to one recipient • check all the required fields (0.5) • check email address format (0.5) • check connection & communication timeout (0.5) • check response codes (1) • show alert messages upon detecting mistake(s) (1) • display all commands & responses (exclude data content) to the console (1) • show the 'successful' alert message and terminate the program (0.5) • successfully receive and correctly display the message by an email user agent (2)
1.5 points	Successfully send a simple text message with multiple To, Cc, & Bcc recipients • correctly parse the list of email addresses and report incorrect address format (0.5) • correctly display the message by the user agent (1)
3.5 points	Successfully send a message with an attachment • successfully receive and correctly display the message with attachment by the user agent (2.5) • successfully download and open the attachment (1)
-6 points	Not using python low-level socket interface
-1 point	Not including the student's info at the beginning of the program
-1 point	Not using the student's CS email address or student's number as the marker
-0.5 points	Incorrect program name

Plagiarism

Plagiarism is a very serious academic offence. Students should understand what constitutes plagiarism, the consequences of committing an offence of plagiarism, and how to avoid it. **Please note that we may request you to explain to us how your program is functioning as well as we may also make use of software tools to detect software plagiarism.**